Wine Quality Prediction Using Machine Learning

Introduction:

The wine industry has seen exponential growth as drinking wine has become commonplace in social gatherings. Nowadays, certifications are being used by market players as a benchmark for the finest quality wine which is a time-intensive process and always requires a human touch as tasting can only be done by us. This makes the process expensive and straining. This can also become a major factor in driving up the cost of the wine. But the price can be based upon a variety of factors, which can be rather abstract as the human side of tasting can sometimes be unscientific and based on emotions the wine provokes. But the quality assessment is usually based on physicochemical tests, which are laboratory-based and consider factors like acidity, pH level, sugar, and other chemical properties.

This process can become more meticulous if human tasting can become analogous to be based on wine's chemical properties.

Problem Statement:

The dataset contains a total of 12 parameters, which were recorded for 1,599 observations. The data is related to red and white variants of the Portuguese "Vinho Verde'' wine. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g., there is no data about grape types, wine brand, wine selling price, etc.). This project's objective is to classify the wines in the good and the poor category using classification models and figure out physicochemical features that are great indicators of good wine. The classes are ordered and not balanced (e.g., there are many more normal wines than excellent or poor ones). Also, we are not sure if all input variables are relevant. We have to predict whether each wine belongs to the “good” category (quality > or = 7).

With this dataset, we will create different classification models to determine how dissimilar independent features predict our target variable, quality. Understanding how each feature will influence the red wine quality will help businesses in the wine industry to enhance their evaluation metrics to improve their production, distribution, marketing, and pricing tactics which can help them gain an edge over their competitors.

Feature Information:

Input variables (based on physicochemical tests):

1. *Alcohol*: the amount of alcohol in wine

2. *Volatile acidity*: are high acetic acid in wine which leads to an unpleasant vinegar taste

3. *Sulfates*: a wine additive that contributes to SO2 levels and acts as an antimicrobial and antioxidant

4. *Citric Acid*: acts as a preservative to increase acidity (small quantities add freshness and flavor to wines)

5. *Total Sulfur Dioxide*: is the amount of free + bound forms of SO2

6. *Density*: sweeter wines have a higher density

7. *Chlorides*: the amount of salt in the wine

8. *Fixed acidity*: these are non-volatile acids that do not evaporate readily

9. *pH*: the level of acidity *Free Sulfur Dioxide*: it prevents microbial

10. *Free Sulfur Dioxide*: it prevents microbial growth and the oxidation of wine

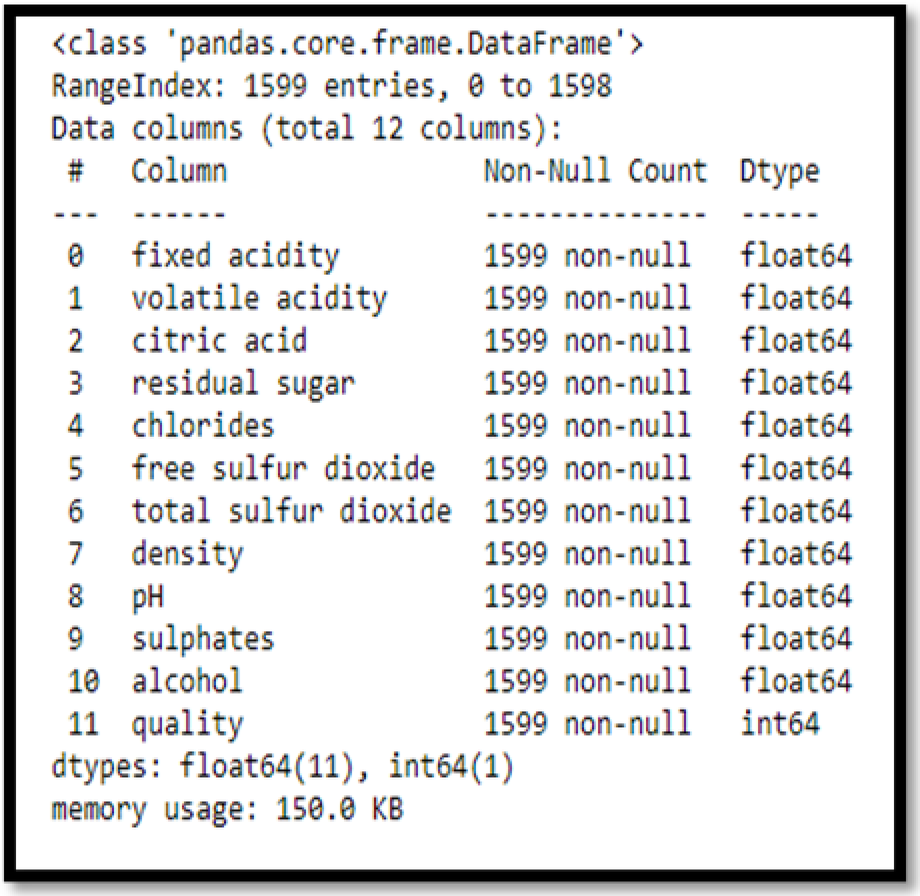
11. *Residual sugar*: is the amount of sugar remaining after fermentation stops. The key is to have a perfect balance between — sweetness and sourness (wines > 45g/L are sweet)

Output variable (based on sensory data):

12. *quality*: score between 0 and 10

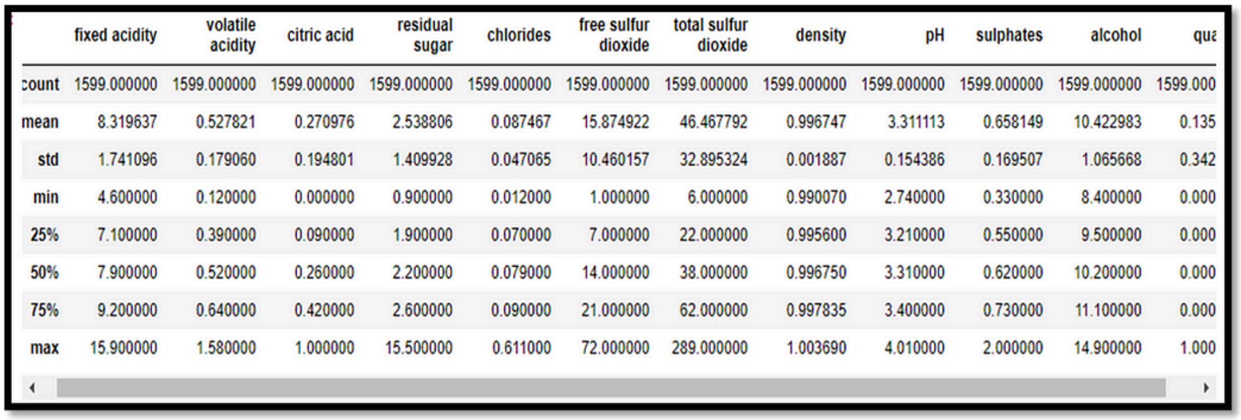
Data Analysis:

The first step in any machine learning project is to prepare and clean the data for the EDA process. I loaded the dataset and saved it as a pandas data frame. Then, I proceeded to the next step where I checked different characteristics of the dataset like data types of the features and the statistical measures of the numerical features using *df.describe()*. Here, we can also use this command to include categorical features using *df.describe(include=”all”)*.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Using df.info()

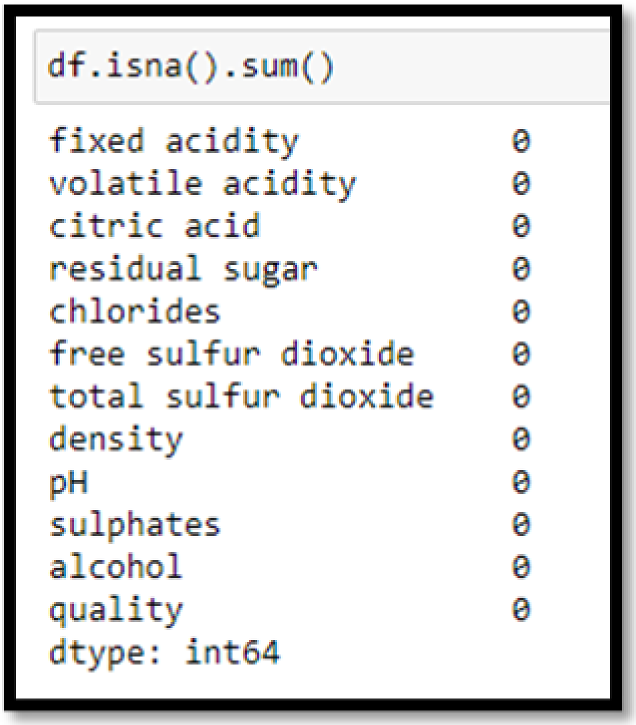


	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	qua
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	0.135
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.342
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	0.000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	0.000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	0.000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	0.000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	1.000

Using df.describe(include="all")

Here, we saw that all the features had the datatype as "float64" except the target columns which belonged to "int64", indicating the fact that "quality" is a whole integer instead of a decimal number.

The next step would be to check if there are any null values present in the dataset. If null values then we can go ahead and impute them with mean for numerical variables and with mode for categorical variables, only if the present quantity is small. If the null values are present in each feature and are a lot, then we can leave them and impute them at a later stage, possibly after Exploratory Data Analysis.



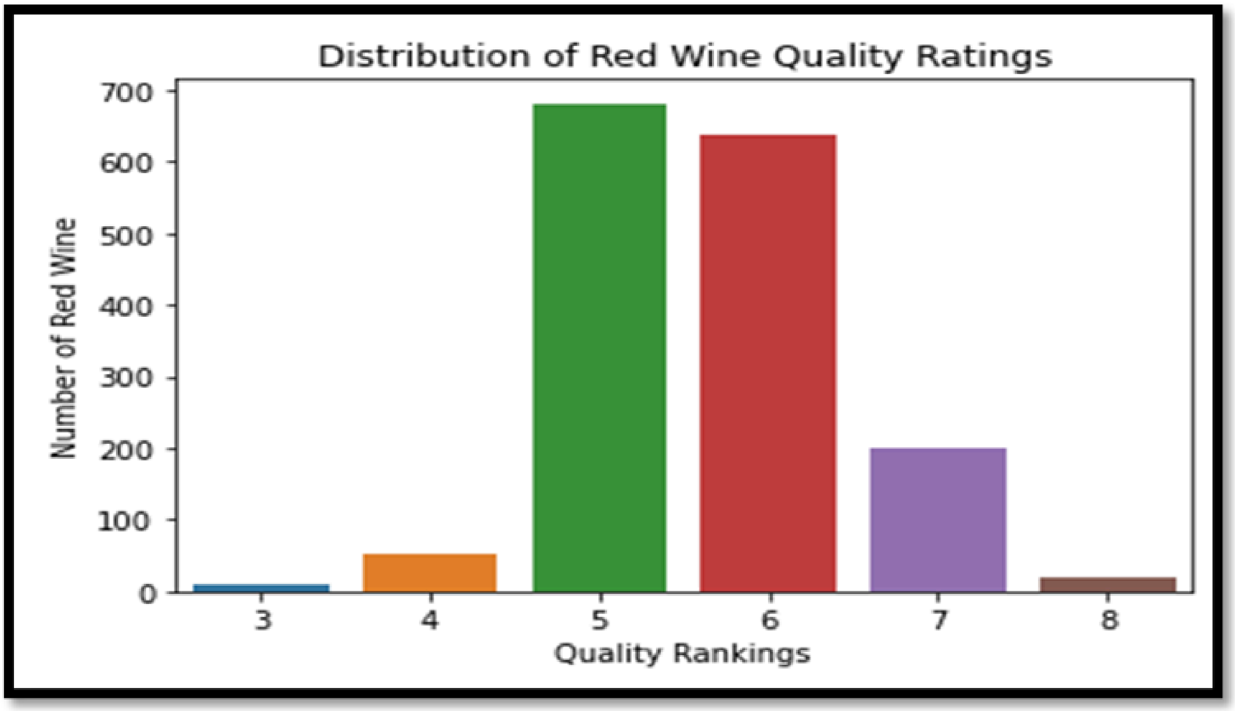
```
df.isna().sum()

fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

We found that there were no Null values present in the dataset.

Exploratory Data Analysis:

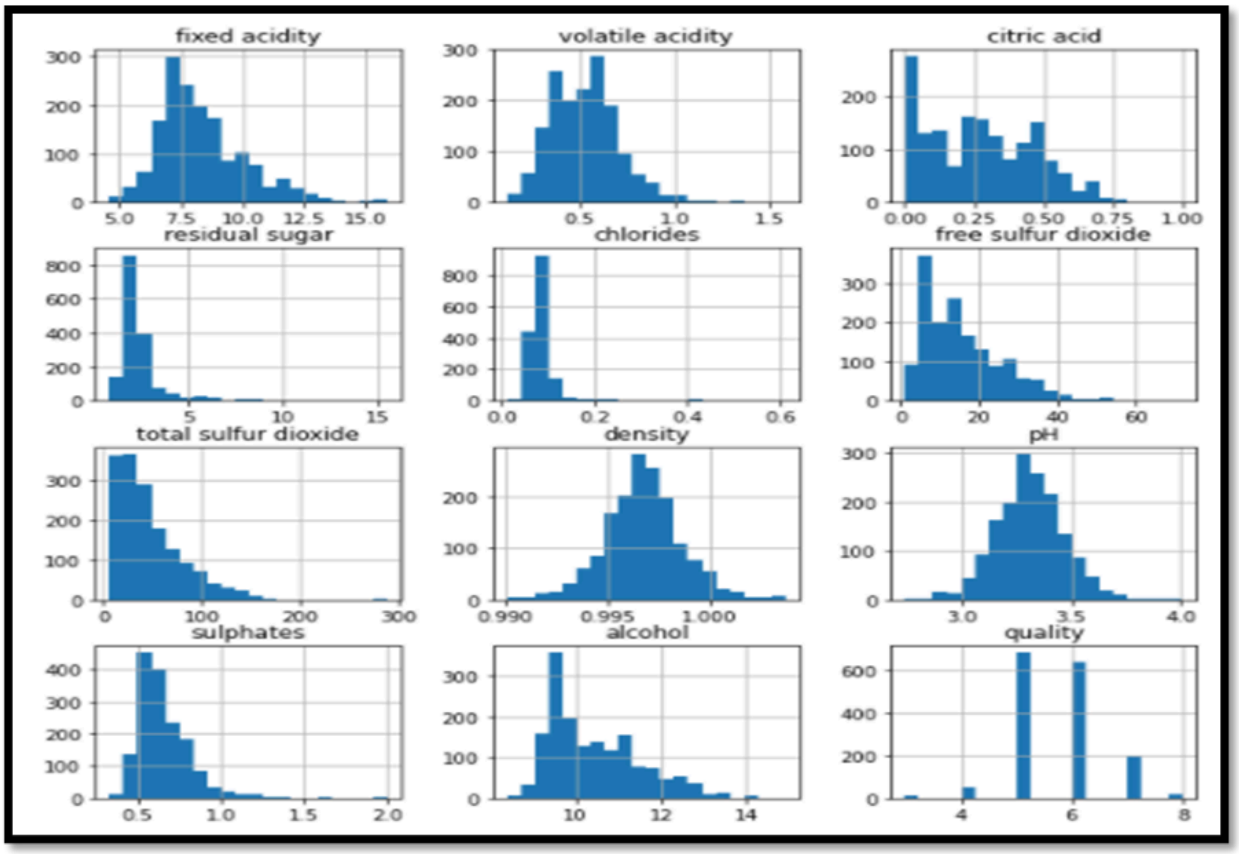
It’s always advisable to proceed with just checking the distribution of all the features including the target column to understand how the data is distributed and the unique values present in each of them. Here, we can infer from the graph below, red wines with 5 and 6 quality bands are present in higher numbers compared to other bands.

7+ quality brand wines are second in number which indicates that there is a class imbalance as 7+ wines have to be considered as good according to the problem statement.



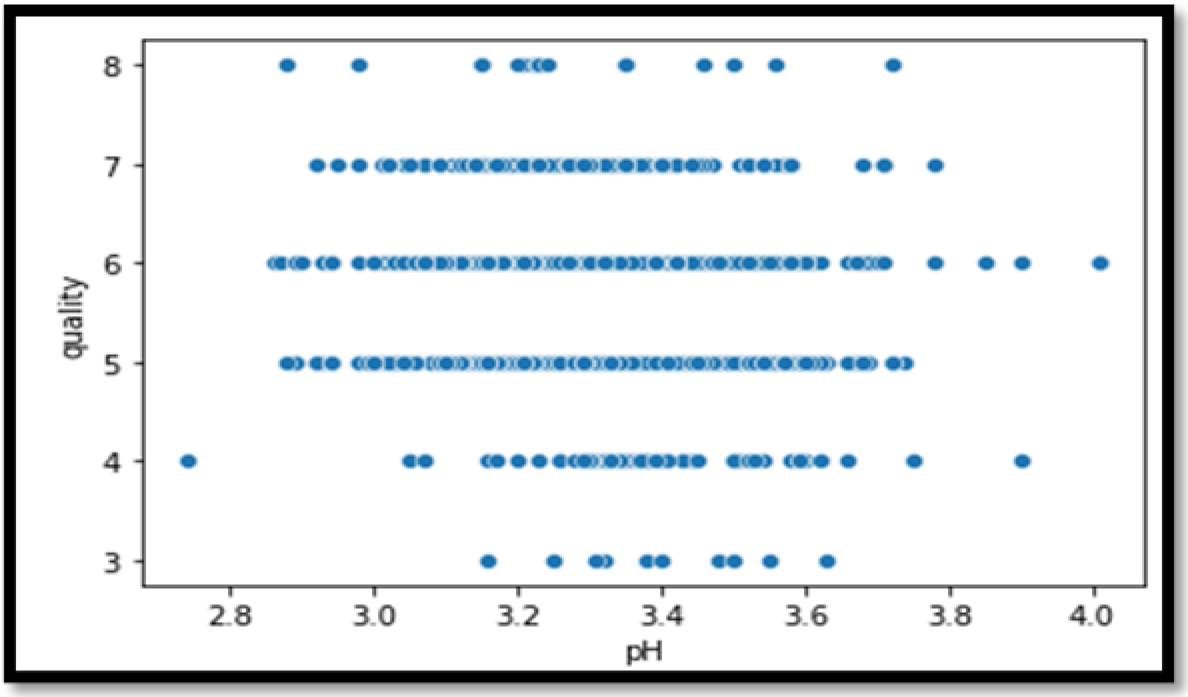
Countplot of the target column

We can then simply call the hist method to plot histograms of each of the numerical features, which indicates that some features are skewed and some follow Gaussian distribution.



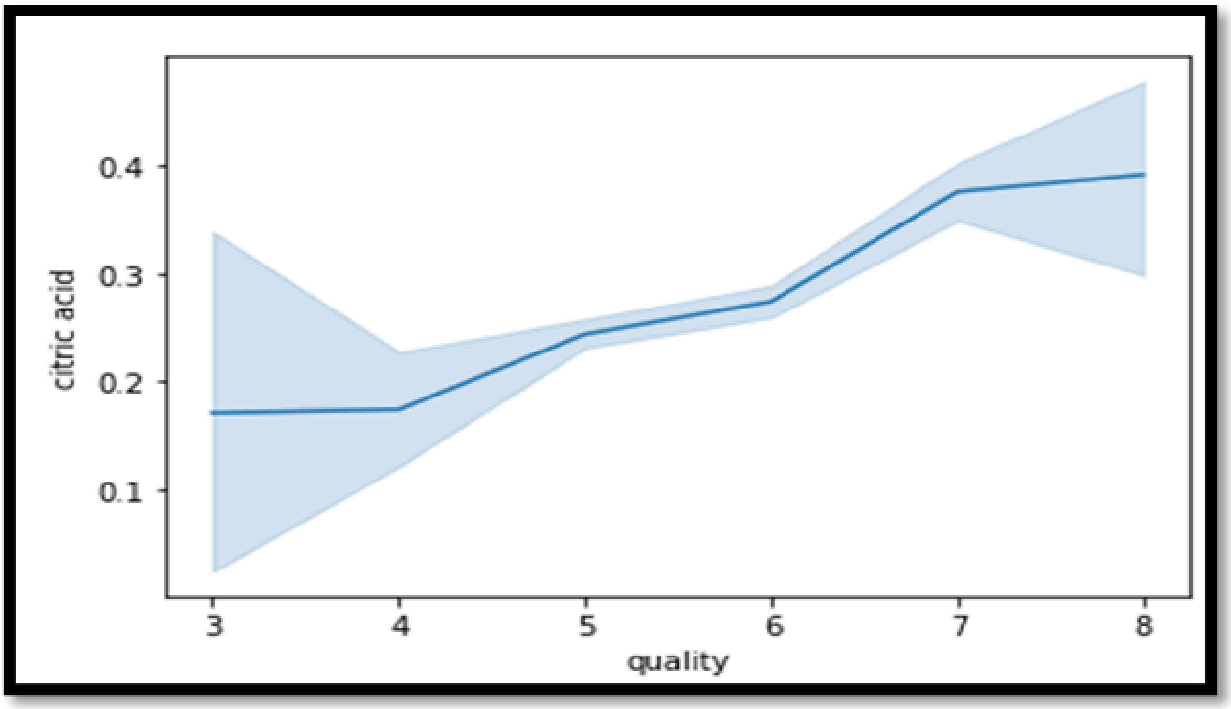
Histograms of various features

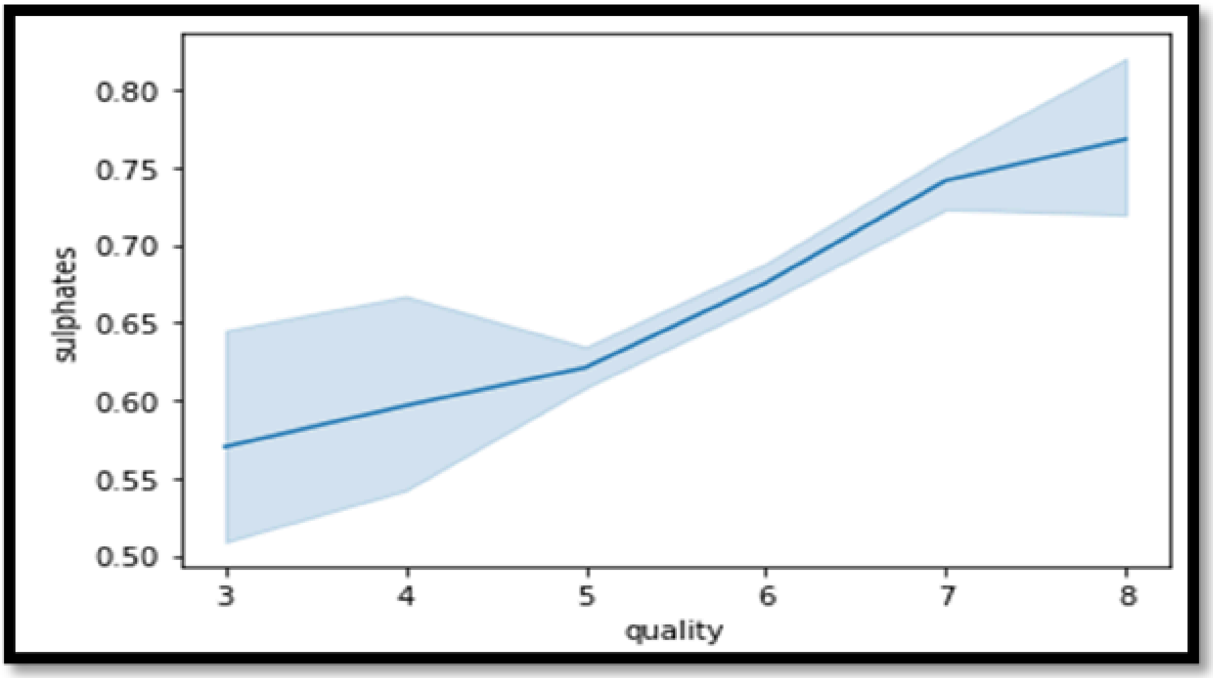
Then I used a scatter plot to check the relationship between the “pH” and “quality” of the wine and found that good quality wines usually have a pH between 2.9 to 3.6 implying that 7+ quality wines are highly acidic in nature, also faintly informing us that they are high in alcohol content.



Quality vs pH

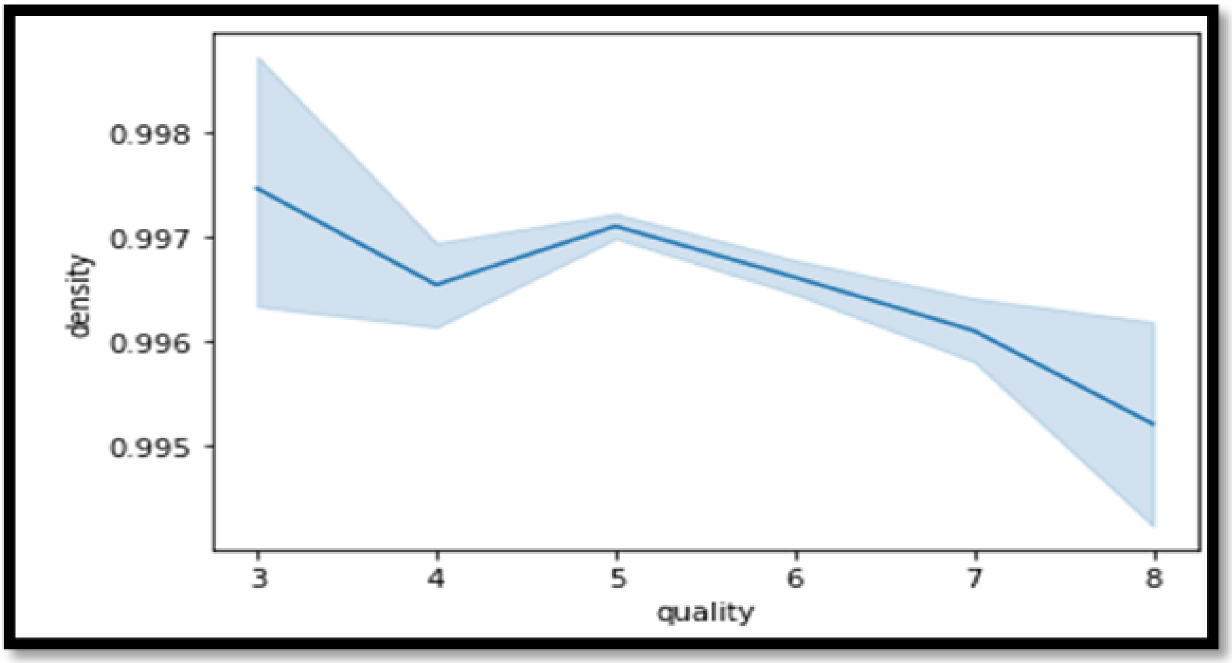
After this, I used a line plot to plot a graph between “quality” and “citric acid” and found out that most of the good wines have a citric acid range from 0.26 to 0.49 with the average being just a little below 0.4. and the second graph between “sulfates” and “quality” indicates that higher quality wines have an average of 0.76 meaning they are highly antimicrobial and have high antioxidant properties.





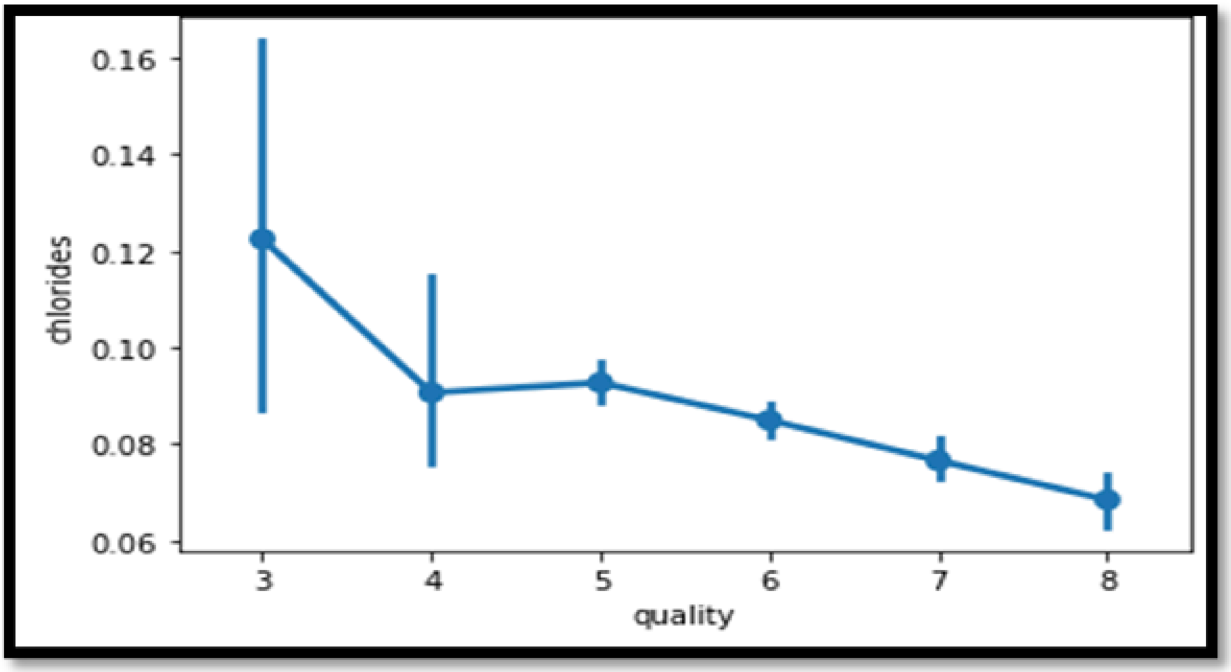
Quality vs sulfates and citric acid

In the next line plot, I found that higher-quality wines have a comparatively lower density than mediocre wines indicating higher sourness as higher-density wines tend to be sweeter as mentioned in the details before.



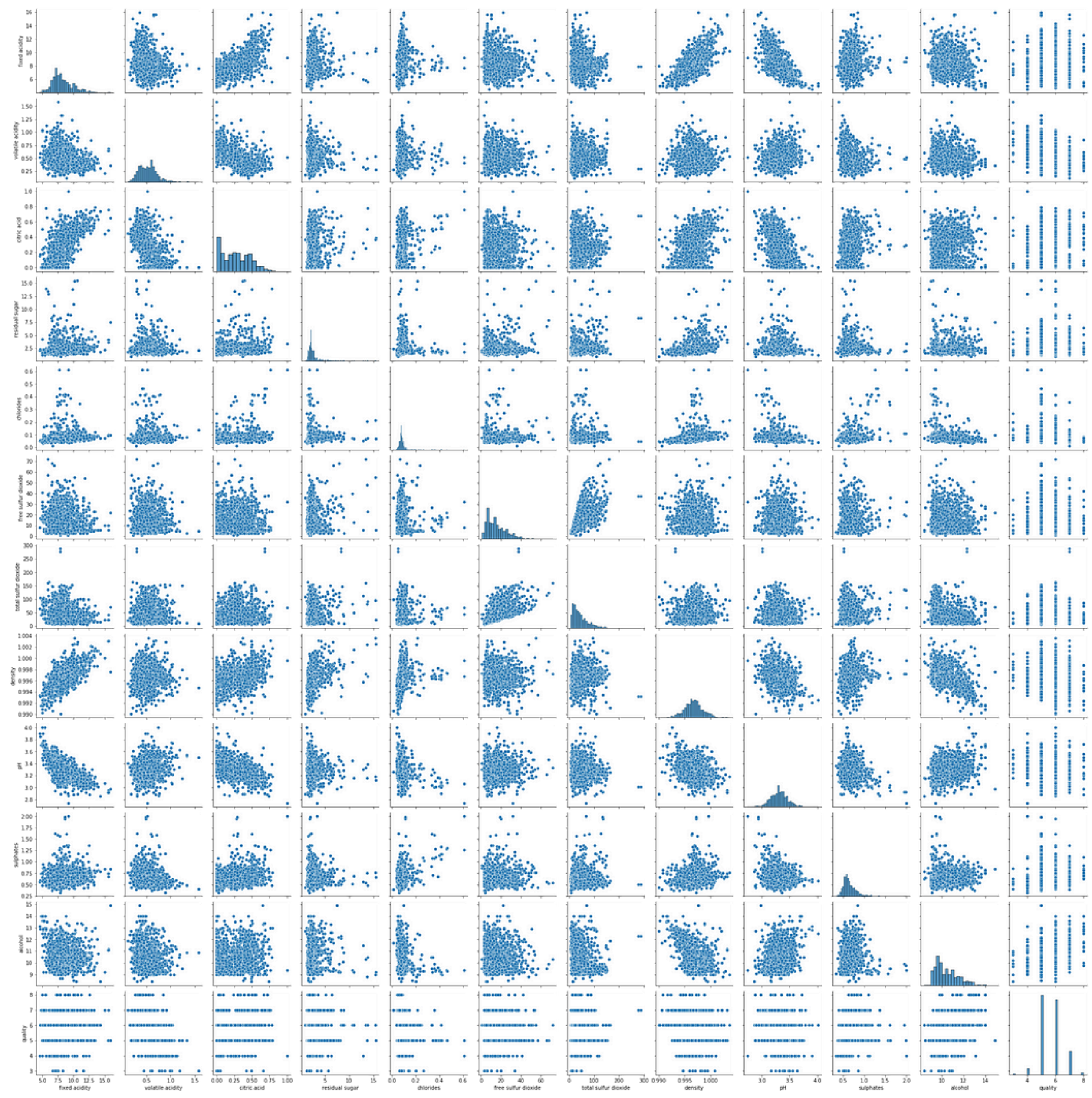
Quality vs Density

I plotted a point plot between “quality” & “chlorides” and saw a downward trend signifying that higher quality wines have lower salt than mediocre wines.



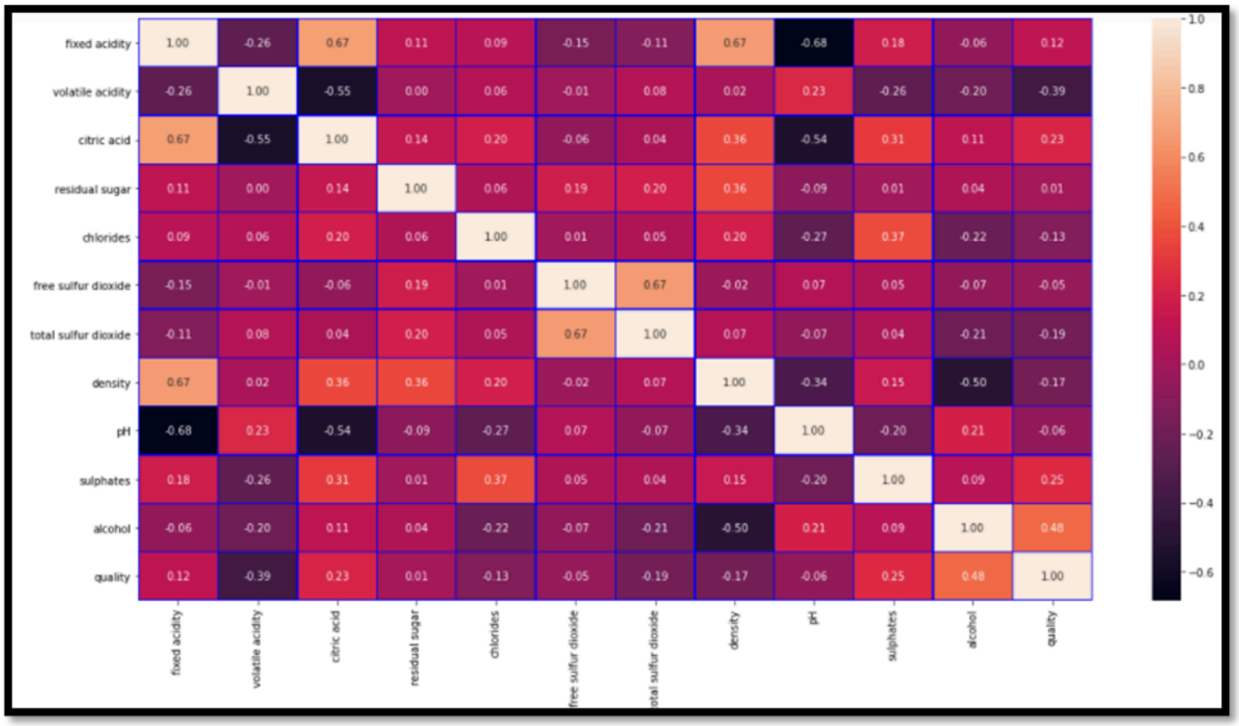
Quality vs chlorides

After that, I used a Pairplot which is a bivariate analysis plot. It plots each feature with each other and the target column. This makes it easier to see the bigger picture and not have to code for each relation.



Using sns.pairplot(df)

Then I plotted the correlation heatmap and also used heatmap to plot the statistics of the dataset.



Heatmap plotting correlation between features and the target

The conclusion is as follows:

· Quality has a (+) positive relationship between alcohol

· Quality has a (-) negative weak relationship between volitile_acidicity

· Quality has almost no relationship between residual_sugar,

free_sulfur_dioxide, and pH (correlation ~ 0)

· Alcohol has a (+) positive relationship between quality and weakly pH

· Alcohol has a (-) negative relationship between density

· Alcohol has almost no relationship between fixed_acidicity, residual_sugar, free_sulfur_dioxide, sulfates.

· Volitile_acidicity has a weak (+) positive relationship between pH.

· Volitile_acidicity has a strong (-) negative relationship between citric_acid

· Volitile_acidicity has a weak (-) negative relationship between fixed_acidicity and sulfates

· Volitile_acidicity has almost no relationship between residual_sugar, chlorides, free_sulfur_dioxide, total_sulfur_dioxide, density

· Density has (+) positive relationship between fixed_acidicity

· Density has (-) relationship between density

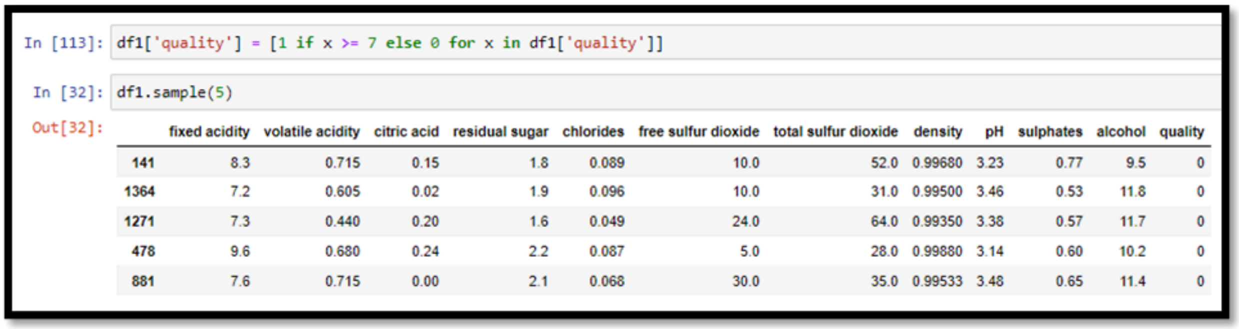
· Density has almost no relationship between volitile_acidicity, free_sulfur_dioxide, total_sulfur_dioxide.

· Citric_acid has (+) positive relationship between fixed_acidicity

· Citric_acid has (-) negative relationship between volitile_acidicity, pH

· Citric_acid has almost no relationship between residual_sugar, free_sulfur_dioxide, total_sulfur_dioxide

After this, we can rewrite the target column and convert it from multiclass classification to a binary classification problem by categorizing wines with quality 7 or above as good wine (1) and other quality wines as poor (0). (This was required by the problem statement given to me during my online course)



```
In [113]: df1['quality'] = [1 if x >= 7 else 0 for x in df1['quality']]
```

```
In [32]: df1.sample(5)
```

Out[32]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
141	8.3	0.715	0.15	1.8	0.089	10.0	52.0	0.99680	3.23	0.77	9.5	0
1364	7.2	0.605	0.02	1.9	0.096	10.0	31.0	0.99500	3.46	0.53	11.8	0
1271	7.3	0.440	0.20	1.6	0.049	24.0	64.0	0.99350	3.38	0.57	11.7	0
478	9.6	0.680	0.24	2.2	0.087	5.0	28.0	0.99880	3.14	0.60	10.2	0
881	7.6	0.715	0.00	2.1	0.068	30.0	35.0	0.99533	3.48	0.65	11.4	0

Also, the heatmap shows which of the features are most important. Importance is in order as follows (by magnitude):

1. Alcohol

2. Volatile acidity

3. Sulphates

4. Citric acid

5. Total sulfur dioxide

6. Density

7. chlorides

8. Fixed acidity

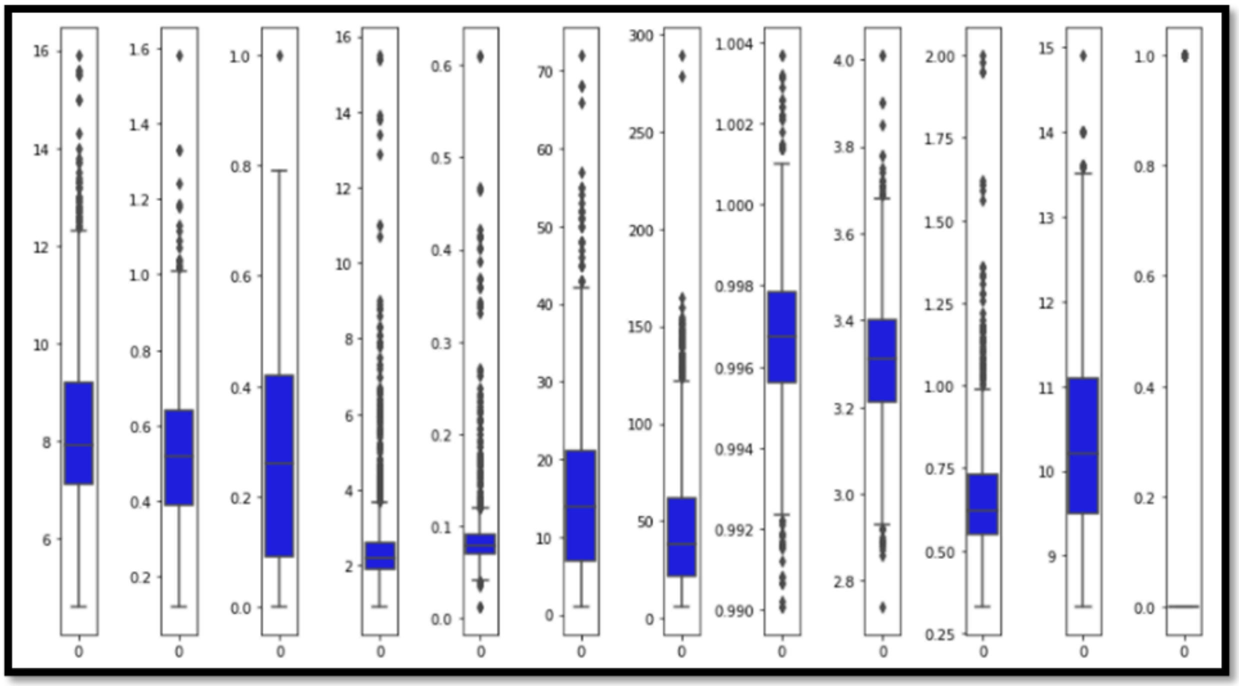
9. pH

10. free sulfur dioxide

11. Residual sugar

Pre-processing Pipeline:

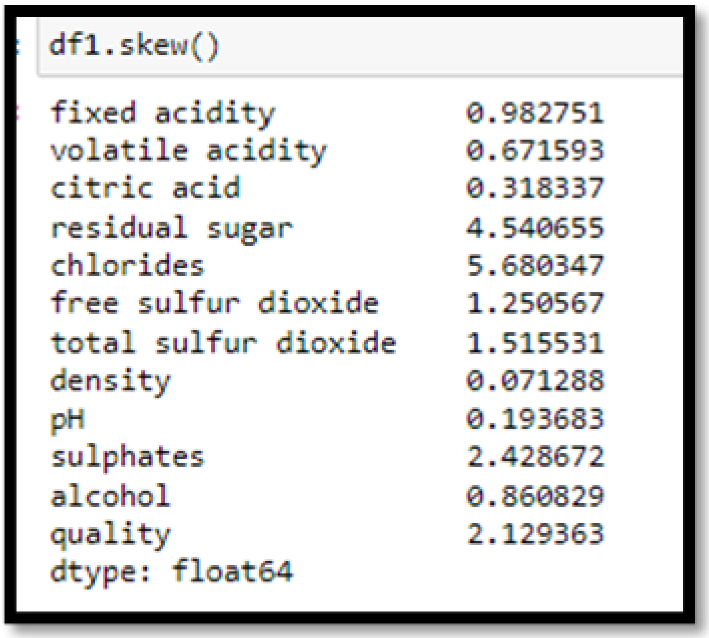
The first step in preprocessing is to check for outliers in the data. This can be best done using boxplots. For this, we will simply use *sns.boxplot* for each column and check which of them have outliers.



Checking for outliers using boxplots

From the plot above, we can see most of the columns have outliers except quality, citric acid, and alcohol which contain relatively fewer outliers.

Secondly, we check the skewness of each numerical column, in this case, our whole dataset is numerical, and if the skewness is found to be outside the range of +/- 0.5, we'll use some kind of transformation like log1p or sqrt.



```
df1.skew()

fixed acidity           0.982751
volatile acidity        0.671593
citric acid             0.318337
residual sugar          4.540655
chlorides               5.680347
free sulfur dioxide     1.250567
total sulfur dioxide    1.515531
density                 0.071288
pH                      0.193683
sulphates               2.428672
alcohol                 0.860829
quality                 2.129363
dtype: float64
```

We can infer from the output above, so columns that need to be transformed are:

1. fixed acidity

2. volatile acidity

3. residual sugar

4. chlorides

5. free sulfur dioxide

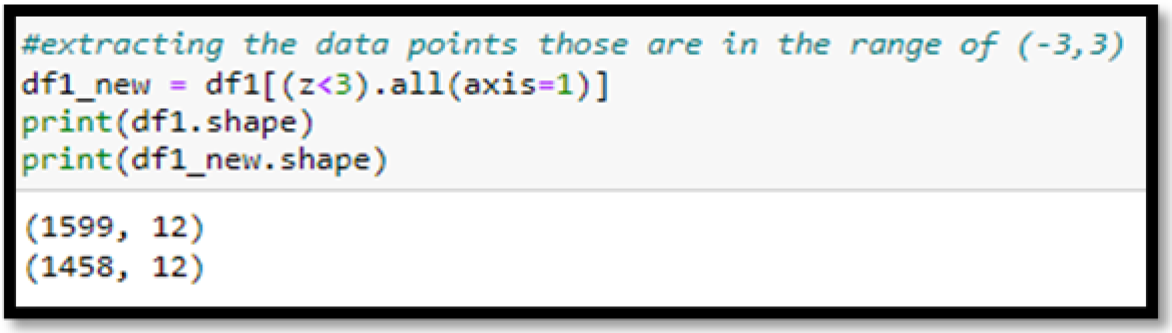
6. total sulfur dioxide

7. sulfates

8. alcohol

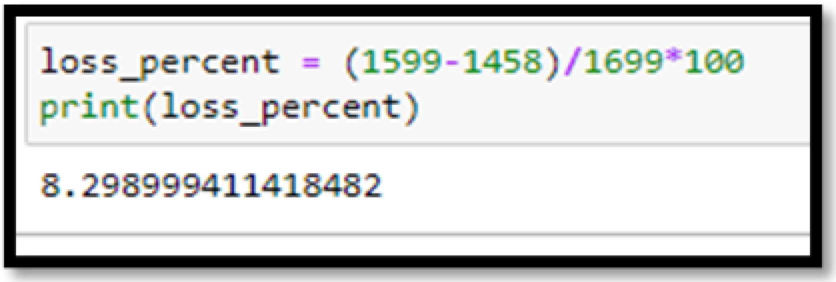
Now we can proceed to remove outliers and skewness from the data, but a very important thing to keep in mind, it is strongly advised to not perform any transformation to remove skewness and scaling (done to make features more or less in the same range) in the target column as it can change the data which is not the right approach, as the model will learn the wrong target values.

Now we can use the z score method to remove outliers from the data and then check the data loss percentage.



```
#extracting the data points those are in the range of (-3,3)
df1_new = df1[(z<3).all(axis=1)]
print(df1.shape)
print(df1_new.shape)
```

```
(1599, 12)
(1458, 12)
```

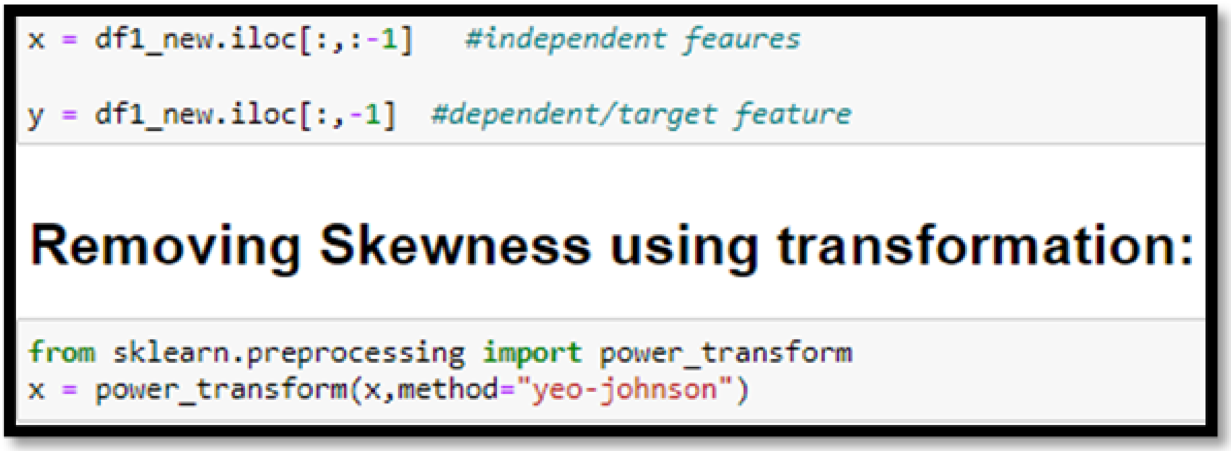


```
loss_percent = (1599-1458)/1699*100
print(loss_percent)
```

```
8.298999411418482
```

We can see that we have lost around 8.3% of the data.

Now we can split the data into Input (x) and Target (y) features. After that, we can simply remove the skewness using power transform in x and then proceed to scale the input features using Standard Scaler.



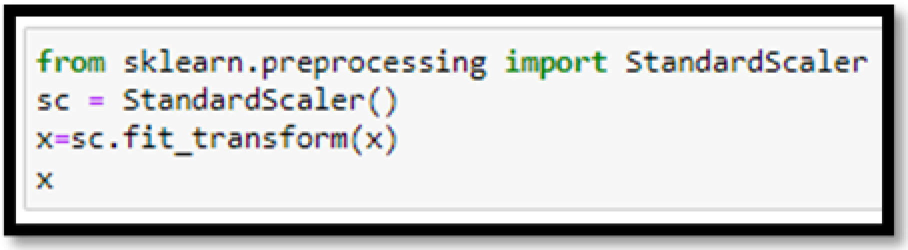
```
x = df1_new.iloc[:,:-1]   #independent feaures

y = df1_new.iloc[:,-1]  #dependent/target feature
```

Removing Skewness using transformation:

```
from sklearn.preprocessing import power_transform
x = power_transform(x,method="yeo-johnson")
```

We can see that the skewness has been greatly reduced for every column except alcohol and residual sugar.

After Scaling:



```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x=sc.fit_transform(x)
x
```

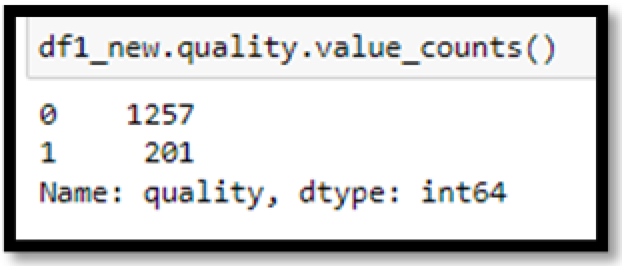
We use the StandardScaler as it's the most commonly used scaling technique.

By Definition: StandardScaler standardizes a feature by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by the standard deviation.

It reduces the data to look more even and reduces the variance greatly, hence, improving the model's learning ability at the model building phase

Handling the class imbalance:

As we know the target variable is hugely imbalanced and to create an accurate model, we need to balance the classes using one of the resampling techniques.



```
df1_new.quality.value_counts()
```

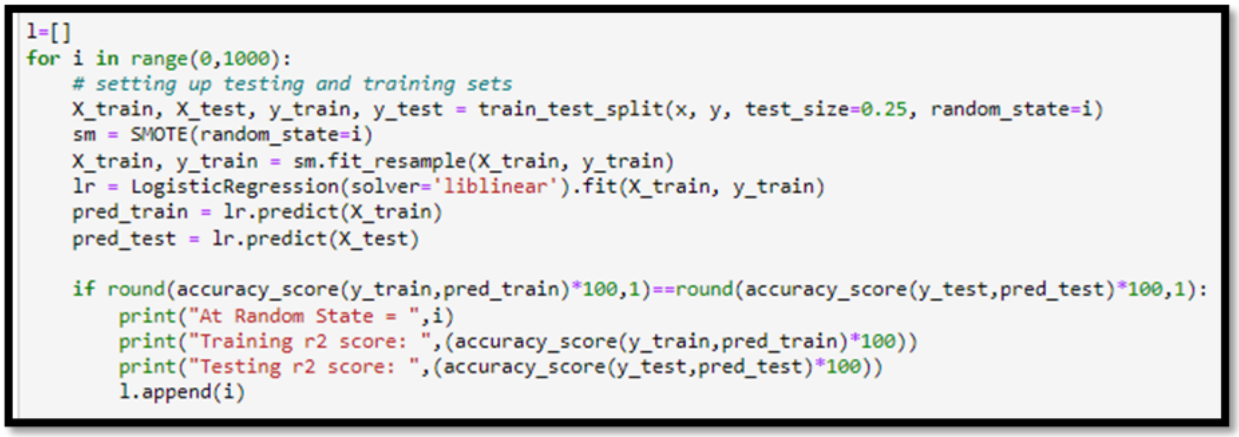
```
0    1257
1     201
Name: quality, dtype: int64
```

Here, SMOTE (Synthetic Minority Oversampling Technique) will be used to upsample the minority class so the dataset can be evenly divided into training and testing datasets.

It is also advisable to split the dataset into testing and training sets before using oversampling techniques.

Oversampling before splitting the data can allow the same observations to be present in both the test and train sets. This can allow our model to simply memorize specific data points and cause overfitting and poor generalization to the test data.

But first, we need to figure out the best random state for our models later. For this, we just use a simple code and find the random state where training accuracy is almost equal to testing accuracy on a resampled X_train and y_train using SMOTE.

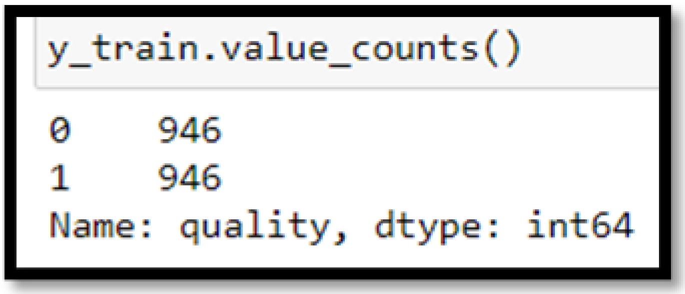


```
l=[]
for i in range(0,1000):
    # setting up testing and training sets
    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=i)
    sm = SMOTE(random_state=i)
    X_train, y_train = sm.fit_resample(X_train, y_train)
    lr = LogisticRegression(solver='liblinear').fit(X_train, y_train)
    pred_train = lr.predict(X_train)
    pred_test = lr.predict(X_test)

    if round(accuracy_score(y_train,pred_train)*100,1)==round(accuracy_score(y_test,pred_test)*100,1):
        print("At Random State = ",i)
        print("Training r2 score: ",(accuracy_score(y_train,pred_train)*100))
        print("Testing r2 score: ",(accuracy_score(y_test,pred_test)*100))
        l.append(i)
```

Discerning the best random state

I found at the random_state in the list above, we have found the best testing and training accuracy for logistic regression. At random state = 653. accuracy touches 83.5%.

Now we can use this random state to train_test_split the x and y to resample X_train and y_train to be used for models in the next step. We can see the minority class has been upsampled and both classes have equal samples in the training target feature.



```
y_train.value_counts()

0    946
1    946
Name: quality, dtype: int64
```

SMOTE magic!

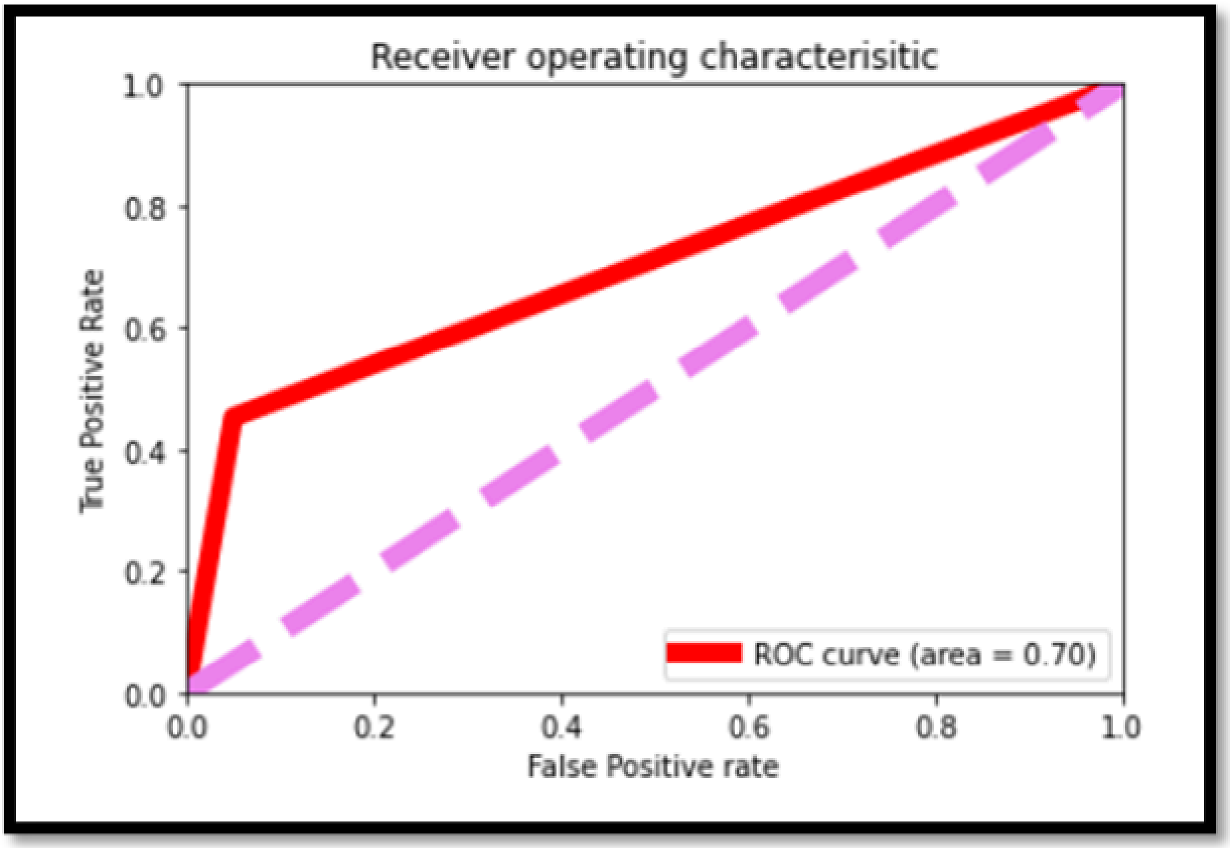
Building Machine Learning Models:

We'll use gridsearchCV for hyperparameter tuning and get the best possible accuracy for our models.

1. Logistic Regression:

With GridSearchCV, I was able to get the best hyperparameters: {'C': 1, 'penalty': 'l2'} which are also the default parameters in the sklearn library for the Log. Reg. model

Then, I trained the model and found the accuracy to be close to 84% on the testing data. To reaffirm this, I used K-fold cross-validation and found that at k=2, the cross_val score was the closest to testing the accuracy of 83.56%. Then I also, auc_roc score and curve area which I found to be 0.70.



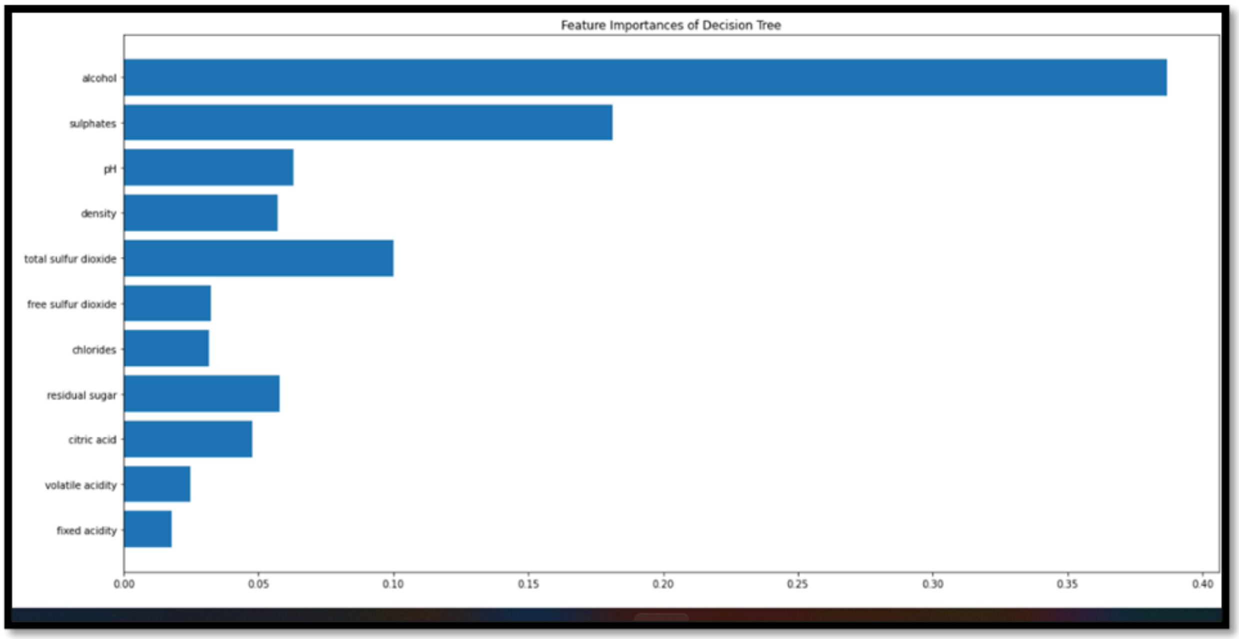
In general, an AUC of 0.5 suggests no discrimination, 0.7 to 0.8 is considered acceptable, 0.8 to 0.9 is considered excellent, and more than 0.9 is considered outstanding.

2. Decision Tree Classifier:

Another point to remember: Best_score (an attribute of gridsearchCV) is the mean of the cross_val_score which is possible with these quantities on a limited dataset. We'll still have to perform cross-validation separately to check how cross-validation impacts our accuracy_score.

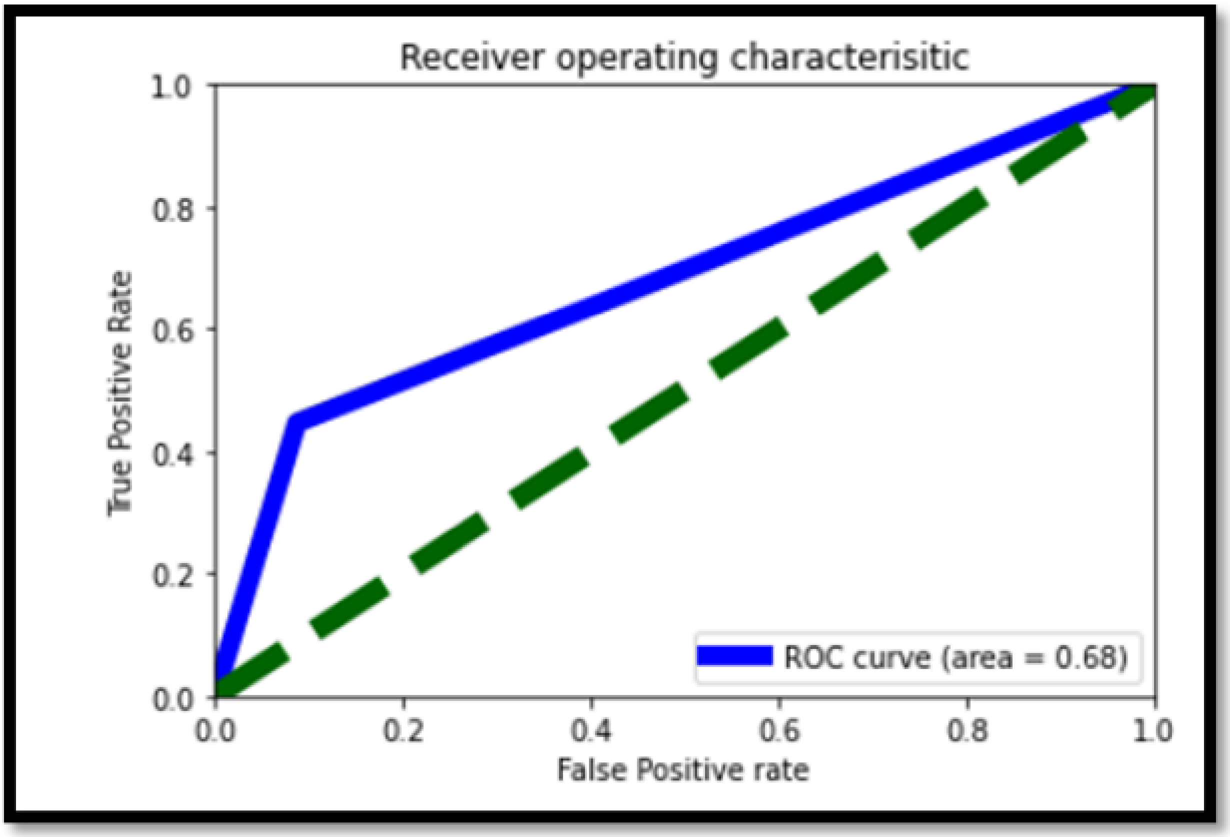
With GridSearchCV, I found :

Best Parameters for Decision Tree: {'criterion':'entropy', 'max_depth': 14, 'min_samples_leaf': 7} Best Score for Decision Tree: 0.8834926248719353



“Alcohol” and “sulfates” were the most important features which had the maximum impact on the model.

At k= 6, Cross Val score: 83.54 and accuracy score is: 84.11



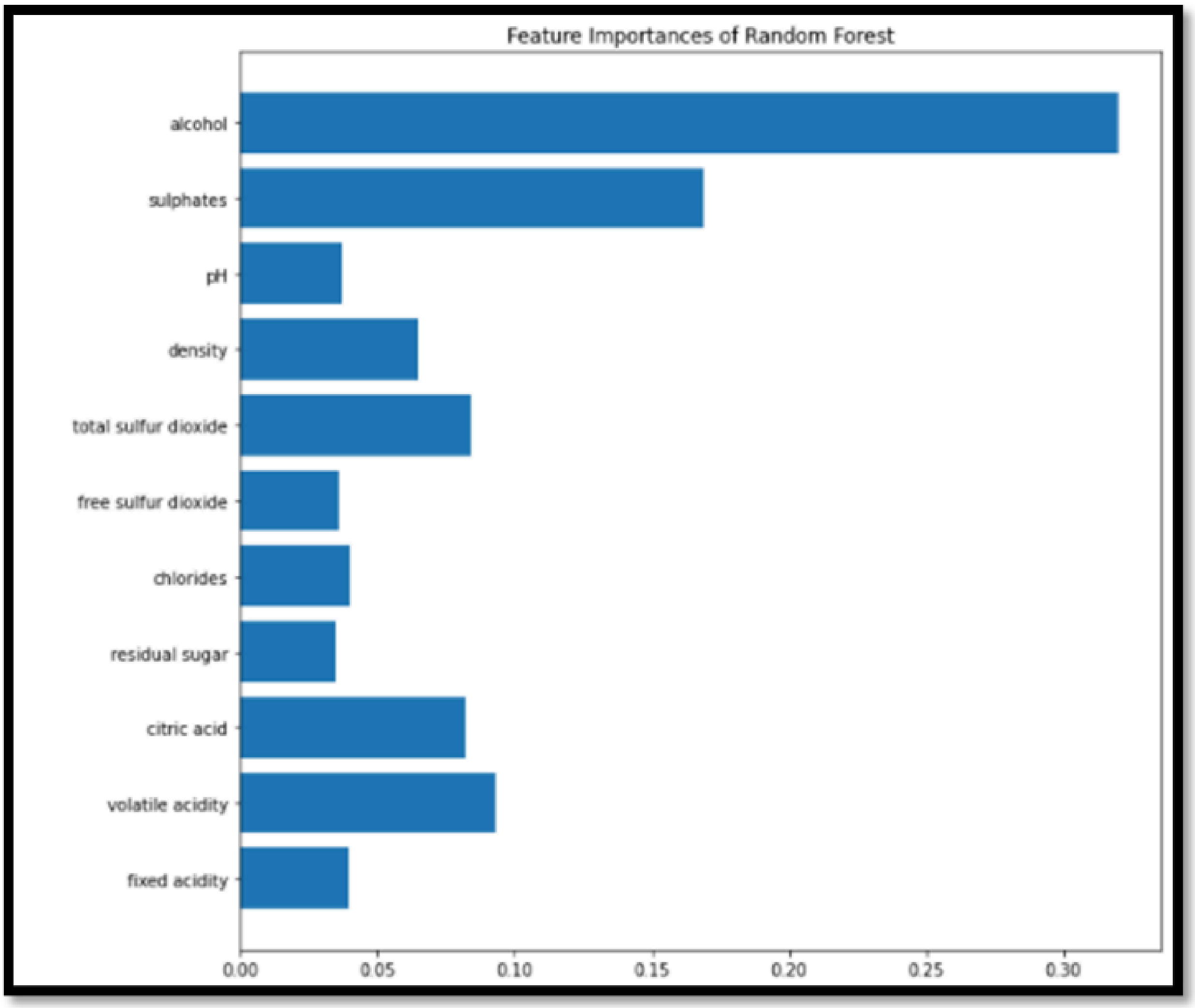
The ROC curve area was found to be 0.68, indicating that even after getting a higher accuracy than Log Reg., the ROC curve area lags and this might not be a better model than the above model.

3. Random Forest Classifier:

With GridSearchCV:

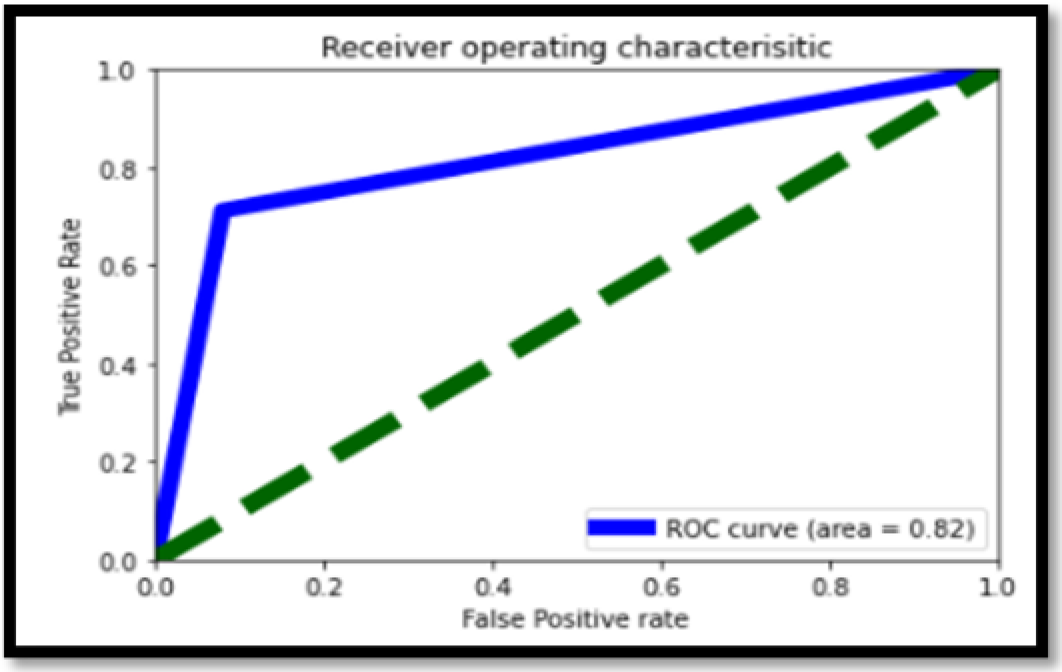
Best Parameters for Random Forest: {‘bootstrap’: False, ‘max_depth’: 9, ‘min_samples_leaf’: 5}

Best Score for Random Forest: 0.9269139545001615



“Alcohol” and “sulfates” were the most important features which had the maximum impact on the model, similar to Decision Tree which makes sense as Random Forest is an ensemble technique where it creates a huge number of decision trees and then averages them to reach a final score.

At k = 9, Cross Val score : 88.68 & accuracy score is : 89.04. This makes sense as ensemble techniques usually have better accuracy than their single counterparts.



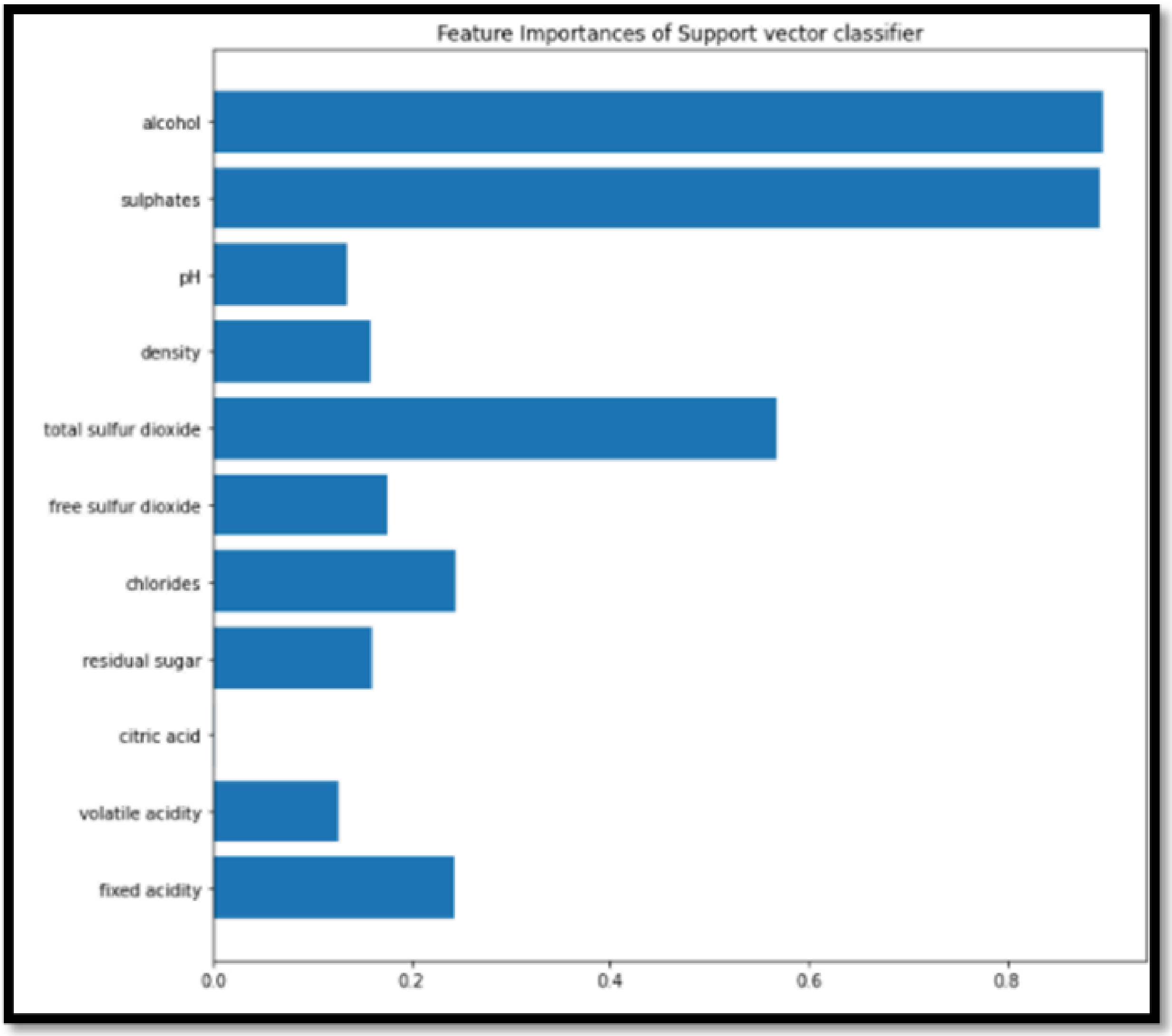
The ROC curve area is found to be 0.82 which makes this model an excellent one.

4. Support Vector Classifier:

With GridSearchCV:

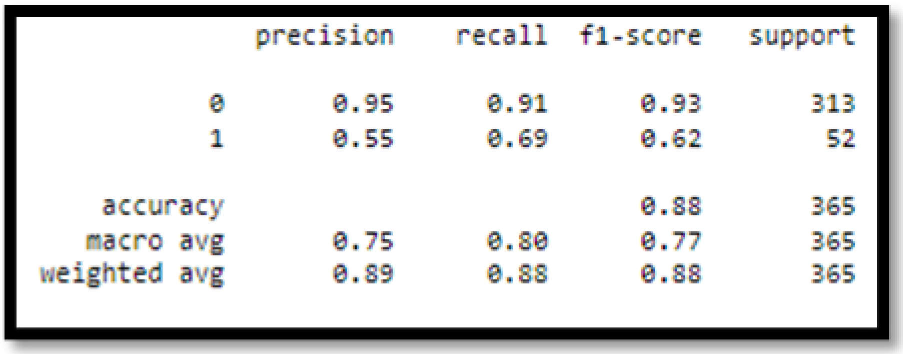
Best Parameters for SVM: {‘C’: 1000.0, ‘kernel’: ‘linear’}

Best Score for SVM: 0.9555162589645348



Alcohol” and “sulfates” were the most important features which had

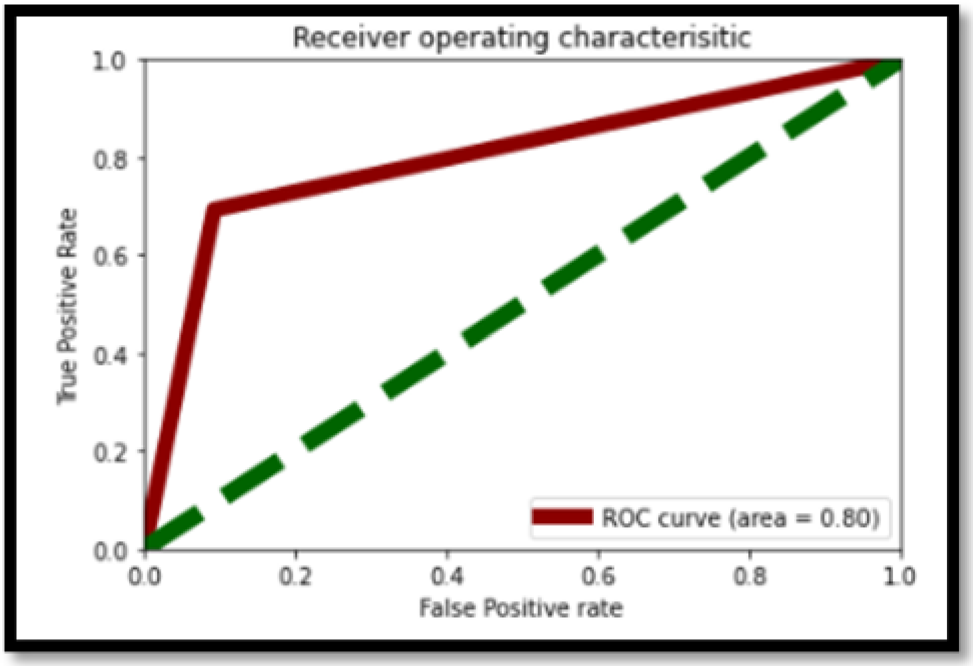
the maximum and equal impact on the model.



```
              precision    recall  f1-score   support

           0       0.95      0.91      0.93       313
           1       0.55      0.69      0.62        52

    accuracy                           0.88       365
   macro avg       0.75      0.80      0.77       365
weighted avg       0.89      0.88      0.88       365
```

Classification report using model predicted target values and actual target values

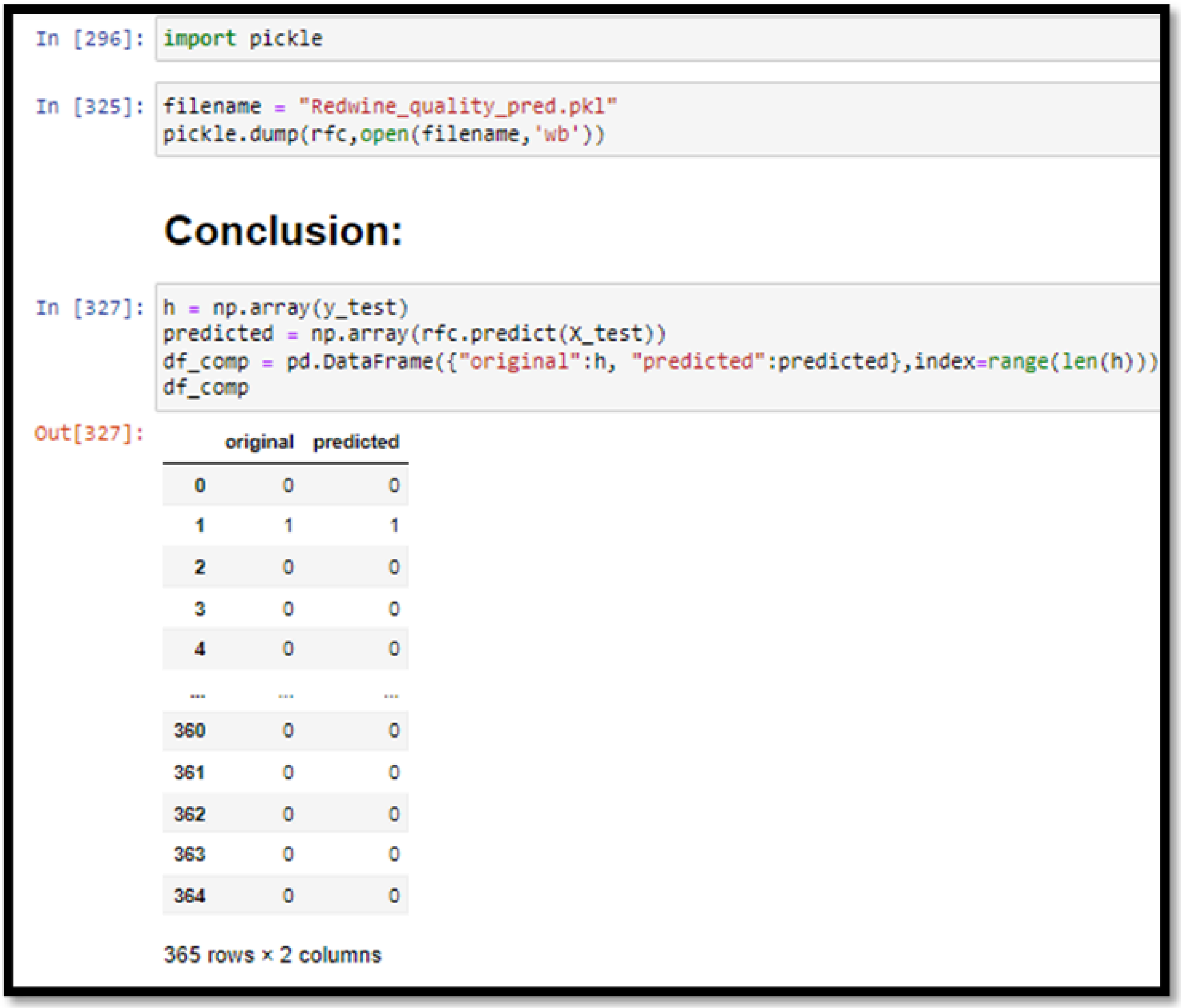
At k= 7, Cross Val score: 86.21 and Accuracy score is: 81.37



The ROC curve area is 0.80 for SVC which indicates it is a good model and will be useful at any test condition.

Conclusion:

We can conclude from these models that Random Forest Classifier performs the best on the testing data with an accuracy of 89%. Now, we can save this model so it can be deployed on a server using the pickle library and dump the model in a local file.



```
In [296]: import pickle

In [325]: filename = "Redwine_quality_pred.pkl"
          pickle.dump(rfc,open(filename,'wb'))
```

Conclusion:

```
In [327]: h = np.array(y_test)
          predicted = np.array(rfc.predict(X_test))
          df_comp = pd.DataFrame({"original":h, "predicted":predicted},index=range(len(h)))
          df_comp

Out[327]:
```

	original	predicted
0	0	0
1	1	1
2	0	0
3	0	0
4	0	0
...	...	...
360	0	0
361	0	0
362	0	0
363	0	0
364	0	0

365 rows × 2 columns

Saving the model and checking the predicted values against the original values

Summary:

The **"Red Wine Prediction Model"** project aims to leverage machine learning techniques to predict the quality of red wine based on various physicochemical properties. The dataset used for this project comprises attributes such as acidity levels, alcohol content, and density, alongside the quality rating provided by experts. Through exploratory data analysis, feature engineering, and model training, the project seeks to develop an accurate predictive model. Initially, the dataset will be preprocessed to handle missing values and normalize features. Next, different machine learning algorithms like Random Forest, Support Vector Machines, and Gradient Boosting will be implemented and evaluated using techniques such as cross-validation and hyperparameter tuning. The model performance will be assessed based on metrics like accuracy, precision, recall, and F1-score. Ultimately, the goal is to deploy a robust prediction model that can assist wine producers in assessing and improving the quality of their products, thereby enhancing customer satisfaction and market competitiveness.

Thank you